Corbin mayes - 1/24/19

Chess Assignment

Minimax and Cutoff Test

To implement the Minimax function, I followed the pseudocode given to us in the textbook and designed a max_value and min_value function. The choose_move would then call whichever method it would need to depending on which side the AI was playing on. By printing the number of calls to those two functions, the number grows exponentially with the depth. A depth of 3 is making over 10,000 calls to the min_value and max_value functions.

Evaluation function

The heuristic or utility function uses the value of each chess piece so queen for example would be 9. It adds all the White pieces and subtracts all the black pieces to get an overall value of the board. So at the start of the game, the board value would be 0 because there is an equal number of pieces on both sides. As you increase the depth, the heuristic value works better because it can see more possible boards which allows for a better decision in the long run.

Iterative Deepening

The Iterative deepening was simple to add as it just needed an extra for-loop to iterate through the different depths up until the max depth.

Alpha-Beta Pruning

To implement the alpha-beta pruning, I first copied over most of the code from minimax AI. Then I altered it to implement the alpha-beta values and search through the possible trees faster. This allowed for a deeper search of the tree in shorter time. By reordering the list, the alpha-beta will be allowed to search deeper because it will cut down on runtime because it isn't searching as far into every tree.

Transposition Table

For the transposition table, I followed the recommendation and used a set that contains the hash(str(board)) of visited boards. This will cut down on the number of calls because it doesn't even run the max_value or min_value functions if the board is already within the transposition table. The number of calls reduces some but not a super large amount.

Related work

I read the paper A general reinforcement learning algorithm that masters chess, shogi and Go through self-play by David Silver and team. The purpose was to design a generalized game AI that functions as well if not better than specialized programs. The focus is on the Deepmind's AlphaZero program and how it differs from other AI and machine learning programs. Their solution to the generality of the program was to replace the "handcrafter knowledge and domain-specific augmentations" for "deep neural networks, a general-purpose reinforcement learning algorithm, and a general-purpose tree search algorithm". They talk about where most programs would use a specialized alpha-beta search with domain-specific enhancements, they instead use a Monte Carlo tree search. The program was trained by playing itself and then would later beat world-renowned programs in chess, shogi, and go.